COMPUTATION: DAY 6

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Review of the fifth day

Languages are either decidable or undecidable. Decidable languages are also called recursive. Among the undecidable languages some are recognizable, also known as recursively enumerable, such as the language of items accepted by a Turing machine itself, hence a bit of an obvious statement,

$$\mathcal{L}(M) = \{\, x \in \Sigma^* \mid \exists t\ M(x;t) = T \,\}$$

Some languages are not Recursively enumerable, and if a language and its complement are recursively enumerable, the language is recursive. Examples are the RE set of Turing machines, known by there index, which accept some input,

$$\neg E_{TM} = \{\, i \in \mathbb{N} \mid \exists x, t\ M_i(x;t) = T \,\}$$

and its co-RE complement,

$$E_{TM} = \{\, i \in \mathbb{N} \mid \forall x, t\ M_i(x;t) \neq T \,\}$$

Among those languages which are not RE, some are the complements of RE language and are called co-RE. Some languages are neither RE nor co-RE. If A is an

RE language, and there are the two reductions

$$A \leq_M B \text{ and } A \leq_M \neg B$$

then B is neither RE nor co-RE. Such a language is the equality and inequality of Turing machines (known by their indices in some enumeration of Turing machines),

$$\neg EQ_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{N} \mid \exists x \, \forall' t \; M_i(x;t) \neq M_j(x;t) \,\}$$

where the notation $\forall' t$ is read *eventually for all* t,

$$\forall' t \equiv \exists t_0 \, \forall t > t_0$$

It follows that equality is also neither RE nor co-RE, and is defined

$$EQ_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{N} \mid \forall x \, \exists' t \; M_i(x;t) = M_j(x;t) \,\}$$

where the notation $\exists'$ is read *exists infinitely often* ,

$$\exists' t \equiv \forall t_0 \, \exists t > t_0$$

Note that eventually for all and exists infinitely often are complements, If something is not eventually always true, it is untrue infinitely often.

2. Complexity classes of Languages

Monday, 3 April 2023

We would like to categorize decidable sets by the difficulty of recognizing them. The focus at the moment is a difficulty related to the number of steps of the most efficient Turing machine that recognizes the language. Several conceptualizations need to be employed to make this work.

3. Algorithms

An algorithm is the idea of a a program on a Turing machine or otherwise reasonably related model of computation. The notion of reasonably related is the strong Turing-Church hypothesis concerning the number of steps taken by the algorithm compared between the two computational models. While what is being counted is the number of steps, this is referred to as "time" as it is considered each step, if the model were physically run, takes a certain unit of time.

For a problem to fit into our schema, there must be an infinite number of problem instances, and each must be stated as a string over the language alphabet, whose string length is important in calculating the run time. The number of steps can depend on exactly the stated instance. The step bound is the maximum number of steps for any instance of a given size. This is a maximum over a finite set. If

the bound is $t(n)$, then the an algorithm A is a Turing machine M_A such that the language is,

$$\mathcal{L}(A) = \{\, x \mid M_A(x; t(|x|)) = T \,\}$$

Here are two algorithms to calculate the greatest common divisor of two positive integers,

```
def gcd_exp(x,y):
    for d in range(min(x,y),1,-1):
        if x%d==0 and y%d==0: return d
    return 1

def gcd_euclid(x,y):
    while y!=0: x,y = y,x%y
    return x
```

They are both algorithms, but they have very different run times. We will associate with the language of greatest common divisors the run time of the Euclidean algorithm, the second code block. In fact, this code is the fastest possible known calculation of the GCD when the run time is stated as a big–oh function class.

4. Asymptotic Algorithmic Complexity

The theory of algorithmic complexity is interested only in the form of the run time for the infinite family of large enough problem instances. We have stated our time bounded Turing machine with a precise function $t(n)$ but in the theory of P and NP we will only care about whether $t(n)$ is of polynomial growth. The notation we will define express this as $t(n) = O(n^c)$ for some integer c.

Definition 4.1. Let $f : X \to Y$ be a function. The class of functions that will be considered *non-strictly smaller* then f asymptotically is defined as,

$$O(f) = \{\, g : X \to Y \mid \exists\, x_o, c > 0\ \forall x \geq x_o,\ cf(x) \geq g(x) \geq 0 \,\}$$

Definition 4.2. Let $f : X \to Y$ be a function. The class of functions that will be considered *strictly smaller* then f asymptotically is defined as,

$$o(f) = \{\, g : X \to Y \mid \forall c > 0\ \exists x_c > 0\ \forall x \geq x_c,\ cf(x) > g(x) \geq 0 \,\}$$

Definition 4.3. A function $f : X \to Y$ is *eventually non-negative* if there exists an x_o such that for all $x \geq x_o,\ f(x) \geq 0$. The function is *eventually positive* if there exists an x_o such that for all $x \geq x_o,\ f(x) > 0$.

The big-oh notation is reflexive, $f = O(f)$, and every function in $O(f)$ is eventually non-negative. Therefore strictly speaking, the notation is restricted to eventually non-negative functions.

It is never true that $f = o(f)$. In general, the notation is restricted to f eventually positive, else the class is the empty set. Every element of $o(f)$ is eventually non-negative.

The little oh is a strict version of the big oh,

$$g = o(f) \implies g = O(f)$$

but not necessarily the converse.

Transitivity works,

$$h = O(g) \wedge g = O(f) \implies h = O(f)$$

and so on.

There is a second, equivalent definition for little oh which is often easier to employ.

Theorem 4.1. Let f be eventually positive and g be eventually non-negative. Then $g = o(f)$ if and only if,

$$\lim_{x\to\infty} g(x)/f(x) \to 0$$

For instance, $n^c = o(a^n)$ for any $a, c > 1$. Exponentials always win.

5. Recognizing Context Free Languages.

Wednesday, 5 April 2023

Lecturer Jamie Deng spoke on the polynomial time recognition of Context Free Languages. A standard way to show this is to first reduce the grammar of the language to Chomsky Normal Form. Then the Cocke–Younger–Kasami algorithm (about 1961) is a dynamic programming algorithm to determine if a given string is in the language in time $O(n^3)$, where n is the total symbol length of the rules.

6. Turing machines that Halt

Friday, 7 April 2023

The machines we are to consider have time bounds and are guaranteed to halt. In our general discussion the notation for a time bounded machine, $M(x;t)$ was the abstract idea of what state the machine provided us by time step t. The undetermined symbol $\perp$ was needed for the case when the machine had neither accepted nor rejected by that time step.

The machines we are considering are bound to halt in some time bound depending on the input x. In particular, a function $t(n)$ such that

$$M(x; t(|x|)) \neq \perp$$

for any x, where $|x|$ is the length of x.

In the light of the undecidability of the halting problem, one might wonder how this bound can be assured. We can modify any multi-tape Turing machine with a timing tape on which is initially the string $1^{t(|x|)}$ and have an explicit right step on this machine for every step the machine makes, and an explicit reject if the head of the timing tape sees a blank.

Monday, 10 April 2023

6.1. **Nondeterministic Turing machines.** The nondeterministic machine is an abstraction where the computation is defined in retrospect but no real step by step computational mechanism. In the case of a time bounded machine, the random steps can be made deterministic by having an additional tape with lists out the result of a sequence of random coin tosses. Without loss of generality, exactly one coin toss, that is, one tape cell, is consumed with each machine step.

In this way we have an actually computing machine, halting in time $t(|x|)$ on input x, exploring a random path path determined by the $t(|x|)$ random bits written onto the random tape. If we write this random tape as a second input, the language A of a non-deterministic machine with time bound $t(n)$ is

$$A = \{\, x \mid \exists y, |y| = t(|x|),\ M(x, y) = T \,\}$$

7. The Classes P and NP

Definition 7.1. A language $A \subseteq \Sigma^*$ is in the class $\mathcal{P}$ of polynomially recognizable languages if there is a Turing machine M_A, in any reasonable time bounded model, and some time bound $t(n) = O(n^c)$ for some c, and

$$A = \{\, x \mid M_A(x; t(|x|)) = T \,\}$$

where $|x|$ is the string length of x.

Definition 7.2. A language $A \subseteq \Sigma^*$ is in the class $\mathcal{NP}$ of nondeterministic polynomially recognizable languages if there is a Turing machine M_A, in any reasonable time bounded model, and some time bound $t(n) = O(n^c)$ for some c, and

$$A = \{\, x \mid \exists y,\ M_A(x, y; t(|x|)) = T \,\}$$

where $|x|$ is the string length of x.

Given the time bound, it is useless to think of a y of length greater than $t(|x|)$. We can add extra letters to y to make it of length exactly $t(|x|)$. If we do this we arrive at the alternative definition of $\mathcal{NP}$ as the class of languages accepted in polynomial time by a nondeterministic Turing machine.

8. Polynomial time Verification

In this definition, of the machine M_A is called the *verifier*, and y is called the *witness* or *certificate*. In this interpretation, y is the solution to a problem, and rather than calculate the solution in polynomial time, we just verify that a given string is a solution.

When y is otherwise looked at as the sequence of random bits which control the computation path in a nondeterministic machine, y is omniscient guidance to the solution. These viewpoints are mathematically equivalent.